Course Name: Database Management Systems
Course Code: TE102
Assignment Type: DBMS Lecture Notes

Database Management Systems

- **Data** are raw facts which can be manipulated. Data is required in the operation of any organization, and the same or similar data may be required for various purposes.
- **Information** is the manipulation of data. In the other words, information is the summarization of data in a presentable form.
- **Data** consists of facts, which become **information** when they are processed and convey meaning to people.
- **A database** is an organized collection of facts. In other words we can say that it is a collection of information arranged and presented to serve an assigned purpose.
- As example of a database is a **dictionary**, where words are arranged alphabetically. Thus information stored in a database is arranged in a particular order.
- A **database management system** (DBMS), or simply a **database system** (DBS), consists of
  - A collection of interrelated and persistent data (usually referred to as the **database** (DB)).
  - A set of application programs used to access, update and manage that data (which form the data management system (MS)).
- The goal of a DBMS is to provide an environment that is both **convenient** and **efficient** to use in
  - Retrieving information from the database.
  - Storing information into the database.
- Databases are usually designed to manage **large** bodies of information. This involves
  - Definition of structures for information storage (data modeling).
  - Provision of mechanisms for the manipulation of information (file and systems structure, query processing).
  - Providing for the safety of information in the database (crash recovery and security).

- Concurrency control if the system is shared by users.

SQL

- SQL is standard language for making queries in relational database packages such as SQL server, Ingress, Sybase, Oracle etc.
- The ORACLE system uses non-procedural Structured Query Language to communicate with its database kernel.
- In 1986, the American National Standard Institute (ANSI) made SQL the standard for all DBMS. It's a powerful query language and all the applications development tools that ORACLE provides are SQL based.
- SQL is a non-procedural language since only the task that has to be achieved is specified, not how to go about doing the job i.e. it provides automatic navigation to the data. The records are processed set at a time rather than just a record at a time.
- SQL does not support any programming language constructs like if..else or while etc, but can be embedded in other programming languages like C, COBOL, PL/ or ADA
- There are two types of SQL: 1) Interactive SQL 2) Embedded SQL
- Interactive SQL is used to operate directly on a database to produce output for our purpose.
- Embedded SQL consists of SQL commands put inside programs that are mostly written in some other high level language. This method can make the program more powerful and efficient.

Features of SQL

- It's an interactive query language that allows the users to use SQL statements to retrieve data and display it o the screen.
- It's a database programming language that allows programmer to embed SQL statements in 3GL programs to access data in a database.
- It's a database administration language that defines the structure of the database and controls the user access to data.
- It's a client/server language that allows application programs on PCs connected via LAN to communicate with the database servers that store shared data.

- It's a database gateway language and often used in gateway that allows one brand of DBMS to communicate with another brand.

SQL*PLUS

- SQL*PLUS is powerful ORACLE support that can take your instructions for oracle. This flexible tool allows both developers and end-user to issue SQL commands directly against the database.
- Database developers use SQL*PLUS to create, fill and monitor an application's database. End-users of the database use it to perform ad hoc queries against the database.
- Basically SQL*PLUS is a command line interpreter. Commands issued in SQL*PLUS are broken into two distinct groups: the ANSI standards SQL commands to create, monitor and manipulate an application's database; and the SQL*PLUS commands provided by ORACLE to enhance the functionality of the ANSI standards SQL commands.
- SQL*PLUS commands control the SQL*PLUS environment, format the output of SQL commands, and control database transaction processing.
- SQL*PLUS provides an open port to the database. An open port gives users, depending upon their access privileges to specific tables in a database unrestrained access to the database's data. When user issues a command, it directly affects the data in the database.
- Several levels of access security prevent SQL*PLUS and user from queuing and modifying every table in the database. Each user runs SQL*PLUS with a specific user

  name with a specific level of database access. Furthermore, a security access is placed on each database table, restricting user from quering and modifying other user's table without proper authorization.

Data Types

When you create table in SQL*PLUS, you must specify the type of data that may appear in each column. Some of the common data types are listed below.

Data Type	Command	Description
Character	CHAR(size)	➢ Fixed length character data. ➢ Size written in the bracket determines the length of data that can be stored. ➢ Default size is 1 and maximum size is 255.
	VARCHAR2 (size)	➢ Variable length character string having maximum length size in bytes. ➢ Maximum size is 2000. ➢ Size must be specified.
Number	NUMBER(p,s)	➢ Used to store variable length numeric data. ➢ P determines the total number of digits possible to the left of decimal point. ➢ S determines the total number of digits possible to the right of decimal point.
	NUMBER (size)	➢ Fixed point number with precision size and scale 0.
Date	DATE	➢ This data type is used to store data and time information. ➢ Default format is DD-MON-YY. ➢ To enter the dates other than standard format, use the appropriate functions.
Long	LONG	➢ Variable length character strings containing up to 2 gigabytes. ➢ Table cannot have more than one Long type of data field. ➢ It cannot be indexed. ➢ It cannot be used with SQL function. ➢ It cannot appear in WHERE, GROUP BY, ORGER BY, clauses.
Raw	RAW(size)	➢ Raw binary data, size byte long. ➢ Maximum size is 255 bytes.

CREATING A TABLE IN THE DATABASE

Sql>**Create table** tablename(column 1 datatype(size) [default <expr>]
[CONSTRAINT constraint name] [column_constraint],
Column 2 datatype(size)…..);

For example:

```
Sql>Create table client_master
    ( c_no varchar2(5), name varchar2(10), address varchar2(20), pincode number(6),
      bal_due number(10,2));
```

INSERTING DATA INTO THE TABLES

The INSERT command with the values clause is used to add new rows to a database table.

```
Sql> INSERT INTO tablename[(column1, column 2…)]
                                    VALUES (value1, value2,..);
```

For example:

```
Sql> INSERT INTO client_master
      (c_no, name, address, pincode, bal_due)
      VALUES ('C001', 'Ajay', 'A-5, Bhandu', 384120, 500 );
```

DATA RETRIVAL USING SQL *PLUS

SQL*PLUS provides a query capability in the form of SELECT statement. One can view the current information in the tables by using this statement.

- The SELECT statement can be used to Display some or all the columns from a specified table.
- Display some or all of the rows from a specified table.
- Display calculated values from the table.
- Display statistical information from the tables, like averages or sums of column values.
- Combine information from two or more tables.

Displaying some or all the Columns from a Table

```
Sql> SELECT column1, column2,……
          FROM tablename;

Sql> SELECT c_no, name

FROM client_master;
Sql> SELECT * FROM tablename;

Sql> SELECT * FROM client_master;
```

Note:

- The SELECT clause followed by the FROM clause are required for any SQL query.
- Not all columns need to be selected.
- Columns are displayed left or right in the order specified.
- SELECT list items must be separated by commas.
- Rows are returned in an arbitrary manner.
- Users may query only tables they have created or tables to which they have granted access.

- If you want to see which tables you have in your account, a special query can be made…

  Sql> **SELECT* FROM tab ;**

Displaying Some Specified Rows from the Table

If you want conditional retrieval of rows i.e. only those rows which satisfy certain condition. You can use WHERE clause in the SELECT statement.

```
sql>SELECT column list
       FROM tablename
            WHERE condition;
```

```
sql>SELECT c_no, name
         FROM client_master
            WHERE bal_due>500;
```

Note:

- Columns specified n the WHERE clause must be part of the table specified in the form clause.
- Columns used in the WHERE clause do not have to be in SELECT list.
- The WHERE clause must specify character data in the same case (upper or lower) that it is in the database.
- The comparison operators that can be used in SQL statements
- < , > , <= .., >= , = , <>

Elimination of duplicates from the select statement:

A table could hold duplicate rows. In such a case, to see only unique rows the syntax is:

```
Sql>SELECT DISTINCT columnname, columnname

          FROM tablename;

Sql>SELECT DISTINCT job

           FROM employee

Sql>SELECT DISTINCT * FROM tablename;

Sql>SELECT DISTINCT * FROM client_master;
```

CRETING TABLE FROM A TABLE

```
Sql>CREATE TABLE tablename[(columnname, columnname)]
          AS SELECT columnname, columnname
                FROM tablename;
```

```
Sql> CREATE TABLE supplier_master
```

```
   (s_no, s_name, address, pincode, bal_due)
  AS SELECT c_no, name, address, pincode, bal_due
     FROM client_master;
```

The source table is a table identified in the select section of this SQL sentence. The target table is one identified in the create section of this SQL sentence.

INSERTING DATA INTO A TABLE FROM ANOTHER TABLE

To insert data one row at a time into a table, it is quite possible to populate a table with data that already exists in another table.

```
Sql> INSERT INTO tablename
        SELECT columnname, columnname,
           FROM tablename;

Sql> INSERT INTO supplier_master
        SELECT c_no, name, address, pincode, bal_due
           FROM client_master;
```

Insertion of data set into a table from another table

```
Sql>INSERT INTO one tablename
          SELECT column1, column2,...

             FROM other table
                [WHERE Condition];

Sql> INSERT INTO supplier_master
        SELECT c_no, name, address, pincode, bal_due
           FROM client_master;
             WHERE  c_no='C001';
```

VIEWING THE STRUCTURE OF THE TABLE

Sometimes need may arise to see the column definition or the integrity constraints specified on them in a particular table. In such conditions DESCRIBE function can be used to view table definition.

```
Sql> DESCRIBE tablename;
```

This displays list of columns and their respective column constraints for the specified table.

MODIFYING THE TABLE DEFINITION

To change the format of an exiting table, we can use ALTER TABLE command.

Sql>**ALTER TABLE** tablename [MODIFY] [ADD]

Adding New Column

Sql> **ALTER TABLE** tablename
    ADD (new columnname datatype(size), new columnname datatype(size));

Sql> **ALTER TABLE** client_master
    ADD (c_fax number(15), city varchar2(6));

Modifying the Existing Columns:

Sql> **ALTER TABLE** tablename
    MODIFY (columnname newdatatype(newsize));

Sql> **ALTER TABLE** client_master
    MODIFY (c_fax varchar2(10)**);**

Note:

- We can increase a CHAR column's width or the number of the digits number of decimal places in the NUMBER column at any time.
- We can change the column from NOT NULL TO NULL by adding the NULL clause to the end of column specification.
- To decrease a column size, the column must be NULL in all exiting rows.
- To add the not NULL feature to a column, the column must have a value for every row in the table. To change the data type of a column, all values in the column must be NULL in all rows.
- To modify more than one column, use command within the parentheses to separate each column from the next.
- You may add a column at any time if NULL is specified. You may not be able to add a column with NOT NULL specified.
- You cannot change the table name and column name.

REMOVING THE TABLE FROM THE DATABASE

Sql>**DROP TABLE** tablename;

Sql>**DROP TABLE** client_master**;**

RENAMING TABLES

Sql>**RENAME TABLE** old table name **TO** new table name;

Sql>**RENAME TABLE** client_master **TO** client_master1;

MODIFYING THE DATA IN EXISTING TABLE:

```
Sql> UPDATE  tablename
         SET column=expression or value
               [WHERE  Condition];
```

```
Sql> UPDATE client_master
         SET name= 'Vijay, address='Mehsana'
            WHERE c_no='c001';
```

REMOVING ROWS FROM A TABLE:

The DELETE command is used to remove rows from a table.

```
sql>DELETE FROM tablename [WHERE condition];
```

```
sql>DELETE FROM client_master WHERE bal_due>500;
```

WHERE clause determines which rows will be removed. If the clause is not specified All the rows from the table will be deleted.

A) Table Name: Client_master

Description: Used to store information about clients.

Column Name	Data Type	Size
Client_no	Varchar2	6
Name	Varchar2	20
Address1	Varchar2	30
Adddress2	Varchar2	30
Ciy	Varchar2	15
Pincode	Number	8
State	Varchar2	15
Bal_due	Number	10,2

B) Table Name: Product_master

Description: Used to store information about products.

Column Name	Data Type	Size
Product_no	Varchar2	6
Description	Varchar2	15
P_percent	Number	4,2
U_measure	Varchar2	10
Qty_on_hand	Number	8
Reorder_lvl	Number	8

Sell_price	Number	8,2
Cost_price	Number	8,2

C) Table Name: Salesman_master

Description: Used to store information about salesman working in the company.

Column Name	Data Type	Size
S_no	Varchar2	6
S_name	Varchar2	20
Address1	Varchar2	30
Address2	Varchar2	30
city	Varchar2	20
Pincode	Number	8
State	Varchar2	20
Sal_amt	Number	8,2
Tgt_to_get	Number	6,2
Ytd_sales	Number	6,2
remarks	Varchar2	60

A) Table Name: Client_master

Client_no	Name	City	Pincode	State	Bal_due
C001	Ivan	Bombay	400054	Maharashtra	15000
C002	Vandana	Madras	780001	Tamil Nadu	0
C003	Pramada	Bombay	400057	Maharashtra	5000
C004	Basu	Bombay	400056	Maharashtra	0
C005	Ravi	Delhi	100001		2000
C006	Rukmani	Bombay	400050	Maharashtra	0

B) Table Name: Product_master

Product_ No	Description	P_ percent	U_ Measure	Qty_on_ hand	Reorder_ lvl	sell_ price	Cost_ price
P001	Floppies	5	Piece	100	20	525	500
P002	Monitor	6	Piece	10	3	12000	11280
P003	Mouse	5	Piece	20	5	1050	1000
P004	Floppies	5	Piece	100	20	525	500
P005	Keyboards	2	Piece	10	3	3150	3050
P006	Cd Drive	2.5	Piece	10	3	5250	5100
P007	1.44 Drive	4	Piece	10	3	8400	8000

C) Table Name: Salesman_master

S_no	S_name	City	Pin	State	Sal_ Amt	Tgt_ to get	Ytd_ Sales	remarks
S001	Kiran	Bombay	40000 2	Maharasht ar	3000	100	50	Good

S002	Manish	Bombay	400001	Maharashta	3000	200	100	Good
S003	Ravi	Bombay	400032	Maharashta	3000	200	100	Good
S004	Ashish	Bombay	400044	Maharashta	3500	200	150	Good

Solve the following queries using the database given above.

A) Retrieving records from table.

1) Find out the names of all clients.
2) Retrieve the entire content of the client_master table.
3) Retrieve the list of names and the cities of all the clients
4) List the various products available from the product_master table.
5) List all the clients who are located in Bombay.
6) Find the names of the salesman who have a salary equal to Rs. 3000

B) Updating records in a table.

1) Change the city of client_no'C002' to 'Bombay'.
2) Change the bal_due of client_no'C001' to Rs.1000
3) Change the cost price of Floppies to Rs. 950.00
4) Change the city of the salesman to Mumbai.

C) Deleting records in a table:

1) Delete all salesman from the salesmane_master whose salaries are equal to Rs. 3500.
2) Delete all products from product_master where the quantity on hand is equal to 100.
3) Delete from client_master where the column state holds the value 'Tamil Nadu'.

E) Altering the table structure:

1) Add a column called 'telephone' of datatype 'number' and size=10 to the client_master table.
2) Change the size of sell_price column in product_master to 10,2.

F) Deleting the table structure along with data:

1) Destroy the table client_master along with its data.

G) Renaming the table:

1) Change the name of the salesman_master table to sman_mast.

<u>COMPUTATION ON TABLE DATA</u>

None of the techniques used till now allows displays of some data from a table after some arithmetic has been done with it.

Arithmetic and logical operators give a new dimension to SQL sentences.

Arithmetic Operators:

Oracle allows arithmetic operator to be used while viewing records from a table or while performing Data Manipulation operations such as Insert, Update and Delete.

+	Addition	*	Multiplication
-	Subtraction	**	Exponentiation
/	Division	( )	Enclosed operation

For example:
Retrieve the content of the column p_no, description and compute 5% of the values contained in the column sell_price for each row from the table product_master.

```
Sql>SELECT p_no, description, sell_price*0.05
    FROM product_master;
```

Renaming Columns used with Expression Lists:

When displaying the result of a query, SQL *PLUS normally uses the selected column's name as the column heading.

These column names may however be short and cryptic; they can be changed for better understanding of the query result by entering an alias, or substitute name, after the column name in the select clause.

```
Sql>SELECT columnname result_columnname, columnname result_columnname
        FROM table name
```

For example:

Retrieve the content of the column p_no, description and compute 5% of the values contained in the column sell_price for each row from the table product_master. Rename sell_price * 0.05 as **Increase.**

```
Sql>SELECT p_no, description, sell_price*0.05  Increase
    FROM product_master;
```

Logical Operators:

Logical operators that can be used in SQL sentence are:

1. AND operator:

The Oracle engine will process all rows in a table and display the result only when all of the conditions specified using the AND operator are satisfied.

```
sql >SELECT column list
        FROM tablename
             WHERE columnname AND columnname;

Sql>SELECT p_no, desc, p_percent
         FROM product_master
           WHERE p_percent>=10 AND p_percent<=20;
```

2. OR operator:

The Oracle engine will process all rows in a table and display the result only when any of the conditions specified using the OR operators are satisfied.

```
sql >SELECT column list
          FROM tablename
              WHERE columnname OR columnname;
sql >SELECT c_no, name, address, pincode
        FROM client_master
           WHERE (pincode=400125 OR pincode=400126);
```

3. NOT operator:

The Oracle engine will process all rows in a table and display the result only when none of the conditions specified using the NOT operator are satisfied.

```
Sql> SELECT c_no, name, address, pincode
        FROM client_master WHERE NOT (city='Bombay' or city='Delhi');
```

Range Searching:

In order to select data that is within a range of values, the **BETWEEN** operator is used. This operator allows the selection of rows that contain values within a specified lower and upper limit.

```
sql >SELECT column list from tablename
       WHERE column BETWEEN min _value AND max_value;
```

sql **>SELECT** c_no, name, address, pincode
FROM client_master
WHERE bal_due **BETWEEN** 100 **AND** 500;

Note:
.BETWEEN is an inclusive operator i.e. if either the min value or the max value is found, as well as any in between, the row is returned.

4. NOT BETWEEN

Rows not having value in the range specified, and also not having value equal; to min or the max value is returned.

sql **>SELECT** column list from tablename
WHERE column **NOT BETWEEN** min _value **AND** max_value;

sql **>SELECT** c_no, name, address, pincode
FROM client_master
WHERE bal_due **NOT BETWEEN** 100 **AND** 500;

Pattern Matching:

1. LIKE

Allows comparison of one string value with another string value, which is not identical. This is achieved by using wildcard characters. Two wildcard characters that are available are:

- The percent sign **(%)** that matches any string.
- The Underscore **(_)** that matches any single character.

sql >**SELECT** column list **FROM** tablename
WHERE column **LIKE** 'pattern';

OR

WHERE column NOT LIKE' pattern';

For example:

Retrieve all information about suppliers whose name begin with the letter 'ja' fro supplier_master.

```
sql >SELECT * FROM supplier_master
          WHERE s_name LIKE 'ja%';
```

2. IN

This operator can be used to select rows that match one of the values included in the list.

```
sql>SELECT columnlist FROM tablename
                 WHERE columnlist IN (list of values);
```

For example:
Retrieve the details from supplier table where supplier name is either Aman or Vimal or Ajay.

```
sql>SELECT s_no, name, city, address, pincode
       FROM supplier_master
          WHERE name IN ('Aman', 'Vimal', ' Ajay' );
```

3. NOT IN

The **NOT IN** predicate is the opposite of the IN predicate. This will select all the rows where values do not match all of the values in the list.

```
sql>SELECT columnlist FROM tablename
                 WHERE columnlist NOT IN (list of values);
```

4. IS NULL

This operator is used to compare the value in the column with NULL and return the row accordingly.

```
sql >SELECT column list FROM tablename
                  WHERE column is NULL;
OR
                  WHERE column is not NULL;
```

ORACLE FUNCTIONS:

Oracle functions serve the purpose of manipulating data items and returning result. Functions are also capable of accepting user-supplied variables or constants and operating on them. Such variables or constants are called as argument. Any number of arguments can be passed to a function in the following format:

Function_name (argument1, argument2, ...).

Oracle functions can be clubbed together depending upon whether they operate on a single row or a group of rows retrived from a table. Accordingly, functions can be classified as follows:

Group Functions (Aggregate Function):
Functions that act on a set of values are called as group functions. For example, SUM, is a function which calculates the total of a set of numbers. A group function returns a single result row a group of queried rows.

Scalar Function (Single Row Function):
Functions that act on only one value at a time are called as scalar functions. For example, LENGTH, is a function, which calculates the length of one particular string value. A single row function returns one result for every row of a queried table or view.

Single row function can be further grouped together by the data type of their arguments and return values. For example, LENGTH, relates to the string Data type. Functions can be classified corresponding to different data types as:

String functions	: Work for String Data type
Numeric functions	: Work for Number Data type
Conversion functions	: Work for Conversion of one type to another.
Date functions	: Work for Date Data type

Aggregate Functions:

AVG	Syntax	AVG([DISTINCT|ALL]n)
	Purpose	Return average value of n ignoring null values.
	Example	Select AVG(sell_price) "Average" from p_master;
	Output	Average 2012.3654
MIN	Syntax	MIN([DISTINCT|ALL]expr)
	Purpose	Return minimum value of 'expr'.
	Example	Select MIN(bal_due) "Min_bal" from c_master;
	Output	Min_bal 0
COUNT	Syntax	MIN([DISTINCT|ALL]expr)
	Purpose	Return the number of rows WHERE 'expr' is not null .
	Example	Select COUNT(p_no) "Products" from P_master;
	Output	Products 9
COUNT(*)	Syntax	COUNT(*)
	Purpose	Return the number of rows in the table, including duplicates and those with nulls..
	Example	Select COUNT(*) "Total" from C_master;
	Output	Total 9
MAX	Syntax	MAX([DISTINCT|ALL]expr)
	Purpose	Return maximum value of 'expr'.
	Example	Select MAX(bal_due) "Maximum" from c_master;
SUM	Syntax	SUM([DISTINCT|ALL]n)
	Purpose	Return Sum of values of 'n'.
	Example	Select SUM(bal_due) "Balance" from c_master;
	Output	Balance 22000

Numeric Functions:

ABS	Syntax	ABS(n)
	Purpose	Return the absolute values of 'n'.
	Example	Select ABS(-15) "Absolute" from dual;
	Output	Absolute 15
POWER	Syntax	POWER(m,n)
	Purpose	Returns m raised to nth power. N must be an integer, else an error is returned.
	Example	Select POWER(3,2) "Raised" from dual;
	Output	Raised 9
ROUND	Syntax	ABS(n[,M])
	Purpose	Returns 'n' rounded to 'm' places right the decimal point. If 'm' is omitted 'n' is rounded to 0 places. 'm' can be negative to round off digit left of the decimal point 'm' must be an integer.
	Example	Select ROUND(15.19,1) "Round" from dual;
	Output	Round 15.2
SQRT	Syntax	SQRT(n)
	Purpose	Returns square root of 'n'. if n<0, NULL. SQRT returns a real result.
	Example	Select SQRT(25) "Square root" from dual;
	Output	Square root 5

String Functions:

LOWER	Syntax	LOWER(char)
	Purpose	Return char, with all letters in lowercase.
	Example	Select LOWER('XYZ') "Lower" from dual;
	Output	Lower xyz
INITCAP	Syntax	INITCAP(char)
	Purpose	Return STRING with first letter in upper case.
	Example	Select INITCAP('COMP DEPT') "Title Case" from dual;
	Output	Title Case Comp Dept
UPPER	Syntax	UPPER(char)
	Purpose	Return char, with all letters in uppercase.
	Example	Select UPPER('xyz') "Upper" from dual;
	Output	Upper XYZ
SUBSTR	Syntax	UPPER(char, M[,n])
	Purpose	Return a portion of char, beginning at character 'm' exceeding up to 'n' characters. If 'n' is omitted, result is returned up to the end char. The first position of char is 1.
	Example	Select SUBSTR('SECURE',3,4) "Substring" from dual;
	Output	Substring CURE
LENGTH	Syntax	LENGTH(char)
	Purpose	Return the length of character.
	Example	Select LENGTH('xyz') "Length" from dual;
	Output	Length 3
LTRIM	Syntax	LTRIM(char[,Set])
	Purpose	Return characters from the left of char with initial.
	Example	Select LTRIM('College','C') "Left" from dual;
	Output	Left ollege

RTRIM	Syntax	RTRIM(char[,Set])
	Purpose	Return char, with final characters removed after the last character not I the set. 'set' is optional, it defaults to spaces.
	Example	Select RTRIM('College','e') "Right" from dual;
	Output	<u>Right</u> Colleg
LPAD	Syntax	LPAD(char1,n,[,char2])
	Purpose	Return 'char1', left padded to length 'n' with the sequence of characters in 'char2', 'char2, defaults to blanks.
	Example	Select LPAD('Page 1',10,'*') "Lpad" from dual;
	Output	<u>Lpad</u> ****Page 1
RPAD	Syntax	RPAD(char1,n,[,char2])
	Purpose	Return 'char1', right- padded to length 'n' with the characters in 'char2', replicated as many times as necessary. If 'char2' is omitted, right-pad is with blanks.
	Example	Select RPAD('page',10,'x') "Rpad" from dual;
	Output	<u>Rpad</u> Pagexxxxxx

<u>Conversion Functions:</u>

TO_NUMBER	Syntax	TO_NUMBER(char)
	Purpose	Converts 'char' , a character value containing a number to a value of number datatype.
	Example	Update P_master set sell_price= sell_price + TO_NUMBER(SUBSTR('$100',2,3)); Here the value 100 will be added to every products selling price in the product_master table.
TO_CHAR	Syntax	TO_CHAR(n[,fmt])
	Purpose	Converts a value of number data type to a value of char data type, using the optional format string. It accepts a number (n) and a numeric format (fmt) in which the number has to appear. If 'fmt' is omitted, 'n' is converted to a char value exactly long enough to hold significant digits.
	Example	Select TO_CHAR(17145,'$099,999')"char" from dual;
	Output	<u>Char</u> $017,145

TO_CHAR	Syntax	TO_CHAR(date[,fmt])
	Purpose	Converts a value of DATE data type to a value of char data type, using the optional format string. It accepts a date (date), as well as format (fmt) in which the date has to appear. 'fmt' must be a date format.. If 'fmt' is omitted, 'date' is converted to a char value in the default date format, i.e. "DD-MON-YY".
	Example	Select TO_CHAR(O_DATE,'Month DD, YYYY) "Format" from s_order where o_no='o42541';
	Output	<u>Format</u> January 26, 20006

Date Conversion Functions:

TO_DATE	Syntax	TO_DATE(char [,fmt])
	Purpose	Converts a character field to a date field.
ADD_MONTHS	Syntax	ADD_MONTHS(D,N)
	Purpose	Returns date after adding the number of months specified with the function
	Example	Select ADD_MONTHS(SYSDATE, 4) from dual;
	Output	<u>ADD_MONTHS</u> 04-AUG-06
LAST_DAY	Syntax	LAST_DAY(d)
	Purpose	Returns the last date of the month specified with the function.
	Example	Select SYSDATE,LAST_DAY(SYSDATE)"Last" from dual;
	Output	<u>SYSDATE</u> <u>Last</u> 04-AUG-06 31-AUG-06
MONTHS_BETWEEN	Syntax	MONTHS_BETWEEN(d1,d2)
	Purpose	Returns number of months between d1 and d2.
	Example	Select MONTHS_BETWEEN('04-AUG-06', '04-JUL-06')"Month" from dual;
	Output	<u>Month</u> 1

NEXT_DAY	Syntax	NEXT_DAY(date,char)
	Purpose	Returns the date of the first weekday named by 'char' that is after the date named by 'date'. 'Char' must be a day of the week.
	Example	select NEXT_DAY('04-feb-06', 'Friday') "Next day" from dual;
	Output	Next day 06-feb-06

The Oracle Table 'DUAL':

Dual is a small Oracle worktable, which consists of only one row and and one column, and contains the value x in that column. Besides arithmetic calculation, it also supports date retrieval and it's formatting.

When an arithmetic exercise is to be performed such as 2*2 or 4/3 etc., there really is no table being referenced; only numeric literals are being used.
To facilitate such calculation via a SELECT, Oracle provides a dummy table called DUAL, against which SELECT statements that are required to manipulate numeric literals can be fired, and output obtained.

Sql>**SELECT** 2*2 **FROM** DUAL;

Output:

```
          2*2
      ___________
           4
```

SYSDATE:

Sysdate is a pseudo column that contains the current date and time. It requires no arguments when selected from the table DUAL and returns the current date.

Sql>**SELECT** sysdate **FROM** DUAL;

Output:

```
           Sysdate
      _____________
       06-jun-06
```

Queries on computation on table data:

1) Find the name of all clients having 'a ' as the second letter in their names
2) Find out the clients who stay in a city whose second letter is 'a'.
3) Find the list of all clients who stay in 'Bombay' or 'Delhi'.
4) Print the list of clients whose bal_due is greater than value 10000.
5) Print the information from sales_order table for orders placed in the month of January.
6) Display the order information for client_no 'c001' and 'c002'.
7) Find products whose selling price is greater than 2000 and less than or equal to 5000.
8) Find products whose selling price is more than 1500. calculate a new selling price as, original selling price * .15. rename the new column in the above query as new_price.
9) List the names, city and state of clients who are not in the state of 'Maharashtra'.
10) Count the total number of orders.
11) Calculate the average price of all products.
12) Determine the maximum and minimum product prices. Rename the output as max_price and min_price respectively.
13) Count the number of products having price greater than or equal to 1500.
14) Find all the products whose qty_on_hand is less than reorder level.

Queries on Date manipulation:

1) Display the order number and day on which clients placed their order.
2) Display the month (in alphabets) and date when the order must be delivered.
3) Display the order_date in the format 'DD-MONTH-YY'. E.G. 12-August-06.
4) Find the date, 15 days after today's date.
5) Find the number of days elapsed between today's date and delivery date of the orders placed by the clients.

DATA CONSTRAINTS

➢ All business application of the world run on business data gathered, stored and analyzed. Business managers determine a set of rules that must be applied to the data being stored to ensure its integrity. Fro instance, no employee in the sales department can have a salary of less than Rs. 1000/-.

- Such limitation have to be enforced on the data, and only that data which satisfies the condition set will actually be stored for analysis.
- Oracle permits data constraints to be attached to table columns via SQL syntax that will check data fro integrity. Once data constraints are part of a table column construction, the oracle engine checks the data being entered into a table column against the data constraints.
- If the data passes this check, it is stored in the table, else the data is rejected.
- Oracle allows programmers to define constraints at:
  - Column Level
  - Table Level

Column Level Constraints:

- If data constraints are defined along with the column definition when creating or altering a table structure, they are *column level* constraints.
- Column level constraints are applied to the current column. The current column is the column that immediately precedes the constraints i.e. they are local to a specific column.
- A column level constraint cannot be applied if the data constraints spans across multiple columns in a table.

Table Level Constraints:

- If data constraints are defined after defining all the table columns when creating or altering a table structure, it is a table level constraint.
- Table level constraint must be applied if the data constraint spans across multiple columns in a table.

INTEGRITY CONSTRAINT

NULL constraints:

- There may be records in a table that do not have values fro every field, either because the information is not available at the time of data entry or because the field is not applicable in every case.
- A NULL value is not equivalent to a value of zero if the data type is number and spaces if the data type is character.

- A NULL value will evaluate to NULL in any expression e.g. NULL multiplied by 10 is NULL
- NULL value can be inserted into columns of any data type.
- If the column has a NULL value, oracle ignores the UNIQUE, FOREIGN KEY, CHECK constraints that may be attached to the column.

NOT NULL constraint defined at the column level:

When a column is defined as not NULL, then that column becomes a mandatory column. It implies that a value must be entered into the column if the record is to be accepted for storage in the table.

Syntax:

```
Columnname datatype(size) NOT NULL
```

```
Sql> Create table client_master
   ( c_no varchar2(5) NOT NULL,
    name varchar2(10) NOT NULL,
    address varchar2(20) NOT NULL
   );
```

The NOT NULL constraint can only be applied at column level.

UNIQUE constraints:

The purpose of a unique key is to ensure that information in the columns is unique, i.e. a value entered in column defined in the unique constraints must not be repeated across the columns. A table may have many unique keys.

UNIQUE constraint defined at the column level:

Syntax:

```
Columnname datatype(size) UNIQUE
```

```
Sql> Create table client_master
   ( c_no varchar2(5) UNIQUE,
   name varchar2(10),
    pincode UNIQUE
   );
```

UNIQUE constraint defined at the Table level:

```
Sql> Create table client_master
     ( c_no varchar2(5), name varchar2(10), address varchar2(20)
       UNIQUE (c_no, pincode));
```

DEFAULT constraint:

- At the time of table creation a 'default value' can be assigned to a column. When the user is loading a record with values and leaves this column empty, the oracle engine will automatically load this column with the default value specified.
- The datatype of default value should match the data type of the column.

Syntax:

```
        Columnname data type(size) DEFAULT (value)
```

Create a sales_order table where the default value for the column dely_type is the character, upper case 'F' (Full).

```
Sql>CRETATE TABLE sales_order
     ( o_no varchar2(6), o_date date, c_no varchar2(6)
      dely_type char(1) DEFAULT 'F',
      b_yn char(1));
```

PRIMARY KEY constraint:

A primary key is one or more columns in a table used to uniquely identify each row in the table. A primary key column in a table has special attributes:

- It defines the column as a mandatory column i.e. the column cannot be left blank. The **NOT NULL** attribute is active.
- The data held across the column must be UNIQUE.

PRIMARY KEY constraint defined at the column level:

```
Sql>CREATE TABLE sales_order_details
      ( detorder_no varchar2(5) PRIMARY KEY,
        P_no varchar2(5) PRIMARY KEY,
        P_rate number (8,2), qty_ordered number(6)
     );
```

PRIMARY KEY constraint defined at the table level:

```
Sql>CREATE TABLE sales_order_details
      ( detorder_no varchar2(5), P_no varchar2(5),
         P_rate number (8,2), qty_ordered number(6)
       PRIMARY KEY (detorder_no, p_no)
     );
```

CHECK constraints

Business rule validations can be applied to a table column by using CHECK constraint. It must be specified as a logical expression that evaluates either to TRUE or FALSE.

```
Sql> Create table client_master
     ( c_no varchar2(5) CHECK (c_no like 'C%'),
      name varchar2(10) CHECK (name=upper(name)),
      address varchar2(20) );
```

Note:

- A CHECK constraint takes longer to execute as compared to other constraints thus, CHECK constraints must be avoided if the constraint can be defined using the Not Null, Primary key, Foreign key constraint.
- A CHECK integrity constraint requires that a condition be true or unknown for the row to be processed.
- The condition must be a Boolean expression that can be evaluated using the a values in the row being inserted or updated.
- The condition cannot contain sub queries or sequences.

FOREIGN KEY constraint:

- Foreign key represent relationships between tables. A foreign key is table whose values are derived from the primary key or unique key of some other table.
- The table in which the foreign key is defined is called a **foreign table** or **Detail table.**
- The table that defines the primary or unique key and is referenced by the foreign key is called the **Primary table** or **Master table.**
- The master table can be referenced in the foreign key definition by using the REFERENCES adverb. If the name of the column is not specified, by default, oracle references the primary key in master table.
- The default behavior of the foreign key can be changed by using the ON DELETE CASCADE OPTION. When the this option is specified in the foreign key definition, if the user deletes a record in the master table, all corresponding records in the detail table along with the record in the master table will be deleted.

Syntax:

```
Columnname datatype(size) REFERENCES tablename [(columnname)]
     [ON DELETE CASCADE]
```

```
Sql>CREATE TABLE sales_order_details
```

```
( detorder_no varchar2(5) REFERENCES sales_order,
  P_no varchar2(5), P_rate number (8,2), qty_ordered number(6)
 PRIMARY KEY (detorder_no, p_no)
);
```

Note:

- Reject an insert or update of a value, if a corresponding value does not currently exit in the master table
- If the ON DELETE CASCADE option is set, a DELETE operation in the master table will trigger the DELETE operation fro corresponding records in the detail table.
- Rejects a DELETE for the Master table if corresponding records in the DETAIL table exit.
- Must references a PRIMARY KEY or UNIQUE columns in primary table.
- Requires that the FOREIGN KEY columns and the CONSTRAINT columns have matching data types.
- May references the same table named in the CREATE TABLE statement.

INTEGRITY CONSTRAINT IN ALTER TABLE COMMAND

Example:
1) Add a primary key data constraint on the column s_no belonging to the table s_master.

```
Sql>ALTER TABLE s_master
      ADD PRIMARY KEY (s_no);
```

2) Add foreign key constraint on the column detorder_no belonging to a table s_order_deatils, which references the table sales_order. Modify column qty_ordered to include the NOT NULL constraint.

```
Sql>ALTER TABLE s_order_datais
     ADD FOREIGN KEY(detorder_no) REFERENCES SALES_ORDER
     MODIFY (qty_ordered number(8) NOT NULL);
```

DROPPING INTEGRITY CONSTRAINTS

Example :

1) Drop the PRIMARY KEY constraint from supplier_master.

```
Sql>ALTER TABLE supplier_master
      DROP PRIMARY KEY;
```

2) Drop FOREIGN KEY constraint on column p_no in table s_order_deatil.

```
Sql> ALTER TABLE s_order_detail
```

DROP FOREIGN KEY;

SUB QUERIES

The SQL sub query is one of the advanced query techniques in SQL. This feature lets you use the result of one query as part of another query.

This plays an important role in SQL mainly due to following reasons.

- SQL sub query is often the most natural way to express a query, because it most closely parallels the English-language description of the query.
- Sub queries lets you break a query down into pieces and then put the pieces together, hence very simple to implement.
- Some queries just can not be written without sub queries.

- Subquery must be enclosed in the parenthesis.

Subqueries can be divided into two broad groups-

- Single row suquery: This is subquery which returns only one value to the outer query.
- Multi row suquery: This is subquery which returns multiple values to the outer query.

IN or NOT IN

List the name, job and salary of people in department 2o who have the same job as people in department 30.

```
Sql>SELECT enmae, job, sal FROM EMP WHERE deptno=20
            and Job IN (SELECT job FROM EMP where deptno=30);
```

ANY or ALL

If a suquery returns more than one row and the comparison other than equality is required (e.g. <,>, <=…etc) then the ANY, ALL operators are used.
If the relation required for the list of values returned by the inner query, then the ANY operator is used.

Example:
Find out the name, job and salary of people in department 20 who earn more than any one in department 30.

```
Sql>SELECT ename, job, sal FROM emp WHERE deptno=20
     AND sal >ANY (SELECT sal FROM emp WHERE deptno=30);
```

If the relation required for the outer column in the WHERE clause is such that it should be TRUE for all the values in the list of values returned by the inner query; then the ALL operator is used.

Example:

Find out the name, job, salary of people who earn salary greater than all the people in department 30.

```
Sql>SELECT ename, job, sal FROM emp WHERE deptno=20
        AND sal >ALL (SELECT sal FROM emp WHERE deptno=30);
```

EXITS:-

The existence of rows in the query may be used to qualify rows of an outer query. The nested SELECT statement will be true if one or more rows are found. If no rows exit in the nested SELECT statement, then that portion of the WHERE clause will be false.

Example:
Find all the departments that have employees who exist in them.

```
Sql>SELECT deptno, dname, loc FROM dept WHERE
     EXITS (SELECT deptno FROM emp WHERE deptno= emp.deptno);
```

Sub queries returning multiple columns:-

Inner queries can return more than one columns. Only one care should be taken to write the query in such a way that the order and the type of the column selected by the inner query.

Example:
Find out who are the highest paid employees in each department.

```
Sql>SELECT deptno, ename, sal from emp WHERE (deptno, sal)
     IN (SELECT deptno, max (sal) FROM emp GROUP BY deptno);
```

Correlated subqueries

Consider the query where we have to find out the employees who earn more than the average salary in their own department.

```
Sql>SELECT deptno, ename, sal FROM emp WHERE
     Sal> (SELECT avg (sal) FROM emp WHERE deptno=deptno);
```

- A correlated suquery refers to a column selected by the main query.
- If a query performs a select from the same table as the main query the main query must define an alias for the table name and the subquery must use the alias to refer to the table name and the suquery must use the alias to refer the column value in the main query's candidate rows.

- HAVING clause can also be used in correlated suquery.

Solve the following queries using sub queries:-

1) Find the product_no and description of non moving products i.e. product not being sold.
2) Find the customer name, address1, address2, city and pincode for the client who has placed order no 'O19001'.
3) Find the client names who have placed orders before the month of may'96.
4) Find out if the product 'floppies' has been ordered by any client and print the client_no, name to whom it was sold.
5) Find the names of clients who have placed orders worth Rs. 10000 or more.

JOINING TABLES

In a real life situation, many times one needs to use queries which request data from two or more tables in the database.

JOINS

- The process of forming rows from two or more tables by comparing the contents of related column is called joining tables.
- The resulting table (containing data from both the original table) is called a join between the tables.
- Joins are foundation of multi-table query processing in SQL.
- Since SQL handles multi-table queries by matching columns, it uses the SELECT command for a search condition that specifies a column match.

Sql>SELECT columns FROM table1, table2,…
WHERE logical express

- The where clause specifies how to join or merge the tables together as well as the search criteria if any.
- Columns from tables may be named in SELECT clause.
- Columns which have the same name in multiple tale named in the from clause must be uniquely identified by specifying tablename.column name.
- Tablename.* is short cut to specify all the columns in the table. More than one pair of columns may be used to specify the join condition between two tables.

Using aliases

When using a join, it is good idea to identify all columns by using their table names. However, typing long table names can be difficult sometimes. So, to overcome this problem one can define short name aliases for tables.

```
Sql> SELECT t1.col1, t1.col2, t2.col3 FROM table t1, table t2
                         WHERE t1.col1= t2.col 1;
```

Here t1 and t2 are used as aliases for table 1 and table 2 respectively.

Equi-joins

- A join that is formed as a result of an exact match between two columns is called an equi-join or a simple join.
- Consider query in which we want to known the name of the employee, his department and its location.

```
Sql > SELECT ename, deptno, loc
                 FROM emp, dept
                  WHERE emp.deptno = dept.deptno;
```

Cartesian join:

If the clause is omitted, a Cartesian join is performed. A Cartesian product matches every row of a one table to every row of the other table.

```
Sql>SELECT ename, loc
                FROM EMP, DEPT
               WHERE job='CLERK';
```

Non-equi join:-

- The join which uses comparison operation other than '=' while defining their joining criteria are called non-equi joins.
- Since this join can result in large number of rows, it is advisable to make a non-equi join in combination with a selection criteria to reduce the rows to a manageable range.

Self-join:-

- A self join is used to match and retrieve rows that have matching value in different columns of the same table.
- Consider that you want to known the number, name and job of each employee, along with the number, name and job of the employee's manager then we can write like this..

Sql > **SELECT** worker.empno, worker.ename, worker.job, manager.empno,
manager.ename, manager.job
FROM emp worker, emp manager
WHERE worker.empno=manager.empno;

- A table can be joined to itself as if it were two tables.
- As with any other join, the join is on columns that contain the same type of information.
- The table must be given s alias to synchronize which columns are coming from which tables.

Outer join:-

- To include rows in a joined result even when they have no match in a joined table, the outer join is used.
- An outer join causes SQL to supply a 'dummy' row for rows not meeting the join condition.
- No data is added or altered in the table, the dummy rows exit only for the purpose of outer join
- Use the outer join operator (+) after the table/ column combination in the WHERE clause that needs the dummy rows.
- Only one of the tables in a join relation can be outer joined.
- A table can outer joined to one other table.
- Rows that do not have a match on the join condition but which are returned because of the outer join may be located by searching of rows with NULL condition.

ORDER BY Clause

The rows displayed from a query do not have any specific order either ascending or descending.

If you want them to be in ascending or descending order in a particular field, then you can control this order for the selected rows.

sql **>SELECT** column list **FROM** tablename **[WHERE** condition**]**

ORDER BY column or expression **[ASC/DESC];**

Example:

If you want to show the employee names in descending order for employee table, then type the following.

sql **>SELECT** e_code, e_name, design, basic

FROM employee

WHERE basic > 2500

ORDER BY e_name DESC;

- The ORDER BY clause must be the last clause to appear in the SQL statement.
- Since ORACLE stores rows in no particular sequence, the ORDER BY clause is the only way to retrieve rows in a specific order.
- Columns that are not in the SELECT list may appear in the ORDER BY clause, provided they are in the table.
- Rows with a NULL value in an ORDER BY column will always be sorted first whether ascending or descending were specified.
- SQL will automatically order the output rows from lowest to highest order, unless you specify the clause DESC.

GROUP BY Clause

This clause tells Oracle to group rows based on distinct values that exist for specified columns. i.e. It groups the selected rows based on the value of expression for each row and returns a single row of summary information for each group.

sql >SELECT grouped -by-columns functions **FROM** table name

GROUP BY column, column.,….;

Example:

Retrieve the product numbers and the total quantity ordered for each product from the s_order_details.

sql >SELECT p_no, sum (qty_ordered) " total qty Ordered"

FROM s_order_details

GROUP BY p_no;

GROUPING FUNCTION

These functions act on a group or set of rows and return one Row of summary information per set.

SUM	Computes the total value of the group
AVG	Computes the average value of the group
MIN	Computes the minimum value of the group
MAX	Computes the maximum value of the group

STDDEV	Computes the standard deviation of the group
VARIANCE	Computes the variance of the group
COUNT	Counts the number of non-NULL values for the Specified group.
COUNT (*)	Counts the number of rows including those having NULL values for the given condition. .

HAVING CLAUSE

The HAVING clause can be used in conjunction with the GROUP BY clause. HAVING imposes a condition on the group by clause, which further filters the groups created by the group by clause.

HAVING and WHERE clauses work in a similar manner. The difference is that WHERE works on rows, while HAVING works on groups. Expression in HAVING clause must be single value per group.

A) Table Name: Client_master
Description: Used to store information about clients.

Column Name	Data Type	Size	Attributes
Client_no	Varchar2	6	Primary key/ first letter must start with 'C'
Name	Varchar2	20	Not Null
Address1	Varchar2	30	
Adddress2	Varchar2	30	
Ciy	Varchar2	15	
Pincode	Number	8	
State	Varchar2	15	
Bal_due	Number	10,2	

B) Table Name: Product_master
Description: Used to store information about products.

Column Name	Data Type	Size	Attributes
Product_no	Varchar2	6	Primary key/ first letter must start with 'P'
Description	Varchar2	15	Not Null
P_percent	Number	4,2	Not Null
U_measure	Varchar2	10	Not Null
Qty_on_hand	Number	8	Not Null
Reorder_lvl	Number	8	Not Null
Sell_price	Number	8,2	Not Null, cannot be 0
Cost_price	Number	8,2	Not Null, can not be 0

Campus Corporate

C) Table Name: Salesman_master
Description: Used to store information about salesman working in the company.

Column Name	Data Type	Size	Attributes
S_no	Varchar2	6	Primary key/ first letter must start with 'S'
S_name	Varchar2	20	Not Null
Address1	Varchar2	30	Not Null
Address2	Varchar2	30	
city	Varchar2	20	
Pincode	Number	8	
State	Varchar2	20	
Sal_amt	Number	8,2	Not Null, cannot be 0
Tgt_to_get	Number	6,2	Not Null, cannot be 0
Ytd_sales	Number	6,2	Not Null
remarks	Varchar2	60	

D) Table Name: Sales_order
Description: Used to store client's orders.

Column Name	Data Type	Size	Attributes
Order_no	Varchar2	6	Primary key/ first letter must start with 'O'
Order_date	Date		
Client_no	Varchar2	6	Foreign key references client_no of client_master table.
Dely_addr	Varchar2	25	
S_no	Varchar2	6	Foreign key references s_no of salesman_master table.
Dely_type	Char	1	Delivery: part(P) / full (F) Default 'F'
Billed_yn	Char	1	
Dely_date	Date		Cannot be less than order_date
Order_status	Varchar2	10	Values('inprocess','fullfiled','backorder','cancelled')

Order_no	Order_date	Client_no	S_no	Dely_type	Billed_yn	Dely_date	Order_status
O1901	12-Jan-06	C001	S001	F	N	20-Jan-06	In Process
O1902	25-Jan-06	C002	S002	P	N	27-Jan-06	Cancelled
O4665	18-feb-06	C003	S003	F	Y	20-Feb-06	Fullfilled
O1903	03-Apr-06	C001	S001	F	Y	07-Apr-06	Fullfield
O4666	20-May-06	C004	S002	P	N	22-May-06	Cancelled
O1908	24-May-06	C005	S003	F	N	26-May-06	In Process

E) Table Name: Sales_order_details

Description: Used to store client's orders with details of each product ordered.

Column Name	Data Type	Size	Attributes
Order_no	Varchar2	6	Primary Key/ Foreign Key references order_no of the sales_order table.
Product_no	Varchar2	6	Primary Key/ Foreign Key references product_no of the Product_master table.
Qty_ordered	Number	8	
Qty_disp	Number	8	
Product_rate	Number	10,2	

Order_no	Product_no	Qty_ordered	Qty_disp	Product_rate
O1901	P001	4	4	525
O1901	P002	2	1	8400
O1901	P003	2	1	5250
O1902	P001	10	0	525
O4665	P002	3	3	3150
O4665	P004	3	1	5250
O4665	P005	10	10	525
O4665	P003	4	4	1050
O1903	P006	2	2	1050
O1903	P004	1	1	12000
O1908	P005	1	0	8400
O1908	P007	10	0	1050

Queries using Having and Group By Clause:

1) Print the description and total qty sold for each product.
2) Find the value of each product sold.
3) Calculate the average qty sold for each client that has a maximum order value of 15000.
4) Find out the sum total of all the billed orders for the month of January

Queries on Joins and Correlation:

1) Find out the products, which have been sold to 'Ivan Bayross'.
2) Find out the products and their quantities that will have to be delivered in current month.
3) Find the product_no and description of constantly sold i.e. rapidly moving products.
4) Find the names of clients who have purchased 'CD Drive'.

5) List the product_no and order_no of customers having qty_ordered less than 5 from the sales_order_details table for the product 'floppies'.
6) Find the products and their quantities for the orders placed b 'Ivan Bayross' and 'Vandana Saitwal'.
7) Find the products and their quantities for the orders placed by client_no 'C001' and 'C002'.

CONSTRUCTING AN ENGLISH SENTENCE WITH DATA FROM TABLE COLUMNS

Example:
Create an English sentence, by joining predetermined string values with column data retrieved from the sales_order table.

The string literals are : Order No.
Was placed by Client No.
On

The columns are: Order_no
Client_no
Order_date

Table Name: Sales_order

Order_no	**Client_no**	**Order_date**
O1001	C001	04-aug-06
O1002	C002	05-aug-06
O1003	C003	06-aug-06

```
SELECT  'order no.'|| order_no || 'was placed by Client No'|| Client_no|| 'on' || order_date
FROM sales_order;
```

Output:

```
'orderno.'||Order_no||'wasplacedbyclientno.'||Client_no||'on'||Order_date
__________________________________________________________________________

Order no. O1001 was placed by Client no. C001 on 04-aug-06
Order no. O1002 was placed by Client no. C002 on 05-aug-06
Order no. O1003 was placed by Client no. C003 on 06-aug-06
```

To avoid a data header that appears meaningless, use an alias as shown below:

```
SELECT  'order no.'|| order_no || 'was placed by Client No'|| Client_no||
          'on' || order_date "Orders placed"
           FROM sales_order;
```

Output:

```
Orders placed
_______________________________________________
Order no. O1001 was placed by Client no. C001 on 04-aug-06
```

USING THE UNION, INTERSECT AND MINUS CLAUSE

Union Clause:

Multiple queries can be put together and their output combined using the union clause. The union clause merges the output of two or more queries into a single set of rows and columns.

Example:
Retrive the names of all the clients and salesman in the city of 'mumbai ' from the tables client_master and salesman_master.

```
SELECT salesman_no, name
       FROM salesman_master
       WHERE city='mumbai'

UNION

SELECT client_no, name
       FROM client_master
       WHERE city='mumbai';
```

Intersect Clause:

Multiple queries can be put together and their output combined using the intersect clause. The intersect clause outputs only rows produced by both the queries intersected i.e. the output in an intersect clause will include only those rows that are retrieved by both the queries.

Example:
Retrieves the salesman name in 'mumbai' whose efforts have resulted into at least one sales transaction.

```
SELECT salesman_no, name FROM salesman_master
              WHERE city='mumbai';

INTERSECT

SELECT salesman_master.salesman_no, name
    FROM salesman_master, sales_order
     WHERE salesman_master.salesman_no = sales_order.salesman_no;
```

The intersect clause picks up records that are common in both queries.

Minus Clause:

Multiple queries can be put together and their output combined using the minus clause. The minus clause outputs the rows produced by the first query, after filtering the rows retrieved by the second query.

Example:
Retrives all the product numbers of non-moving items from the product_master table.

Table name: sales_order_deatils

Order No	Product No
O111	P112
O112	P111
O113	P113
O114	P114

Table name: product_master

Product No	Description
P112	Monitors
P113	CD Drives
P116	Keyboards
P117	HDD

SELECT product no **FROM** product_master
MINUS
SELECT product no **FROM** sales_order_details;

The minus clause picks up records in the first query after filtering the records retrived by the second query. Thus, the output after applying the MINUS clause will be:

Output:

```
Product No
_______________

    P113
    P114
```

Queries on Constructing Sentences with data:

1) Print information from product-master, sales_order_detail tables in the following format for all records:-
{description} worth Rs. {total sales for the product} was sold.

2) Print information from product-master, sales_order_detail tables in the following format for all records:-

{description} worth Rs. {total sales for the product} was ordered in the month of {order- date in month format}.

3) print information from product-master, sales_order, client_master tables in the following format for all the records:-

{cust_name} has placed order {order_no}on {order_date}.

SQL PERFORMANCE TUNING

INDEXES

- An index is an ordered list of the content of a column of a table.
- Indexing a table is an 'access strategy', that is a way to sort and search records in the table. Indexes are essential to improve the speed with which the record can be located and retrieved from a table.
- When data is inserted in the table, the oracle engine inserts the data value in the index.
- For every data value held in the index the oracle engine inserts a unique **ROWID** value.
- The Rowid indicates exactly where the record is stored in table.
- The Rowid format used by oracle is as follows:

BBBBBBB.RRRR.FFFF

- **FFFF** is a unique number given by the oracle engine to each data file. Data file are file used by oracle engine to store user data.
- **BBBBBBB** is the block number in which the record is stored. The data file is divided into data blocks and each block is given a unique number.
- **RRRR** is unique record number. Each data block can store one or more records. Thus each record in the data block is given a unique record number.

Creating INDEX:

```
Sql>CREATE INDEX index_name
   ON tablename (columnname);
```

Example:

```
Sql>CREATE INDEX idx_c_no
          ON client_master(c_no);
```

Creation of Unique Index:

An unique index can also be created on one or more columns. If an index is created on a single column it is called simple Unique Index.

```
Sql>CREATE UNIQUE INDEX indexname
             ON tablemname (columnname);
```

When the user defines a primary key or a unique key constraint, the oracle engine automatically creates a unique index on the primary key or unique key column.

If an index is created on more than one column it is called Composite Unique Index.
The syntax for creating a composite unique index is:

```
Sql>CREATE UNIQUE INDEX indexname
             ON tablemname (columnname, columnname);
```

Dropping INDEX:

```
Sql>DROP INDEX index_name;
```

- For each row returned by query, the ROWNUM column returns a number indicating the record in which the oracle engine selects the row from table. The first row selected has a ROWNUM of 1, the second has 2 etc.
- The order in which data is retrieved is dependent upon the indexes created on the table.
- If an index is created on the column used I the order by clause, the oracle engine uses the index to retrieve data in sorted order. Thus the ROWNUM will be in the order of the rows retrieved from the index.
- If an index is not created on the column used in the order by clause, the oracle engine will retrieve data from the table in the order of data insertion and thus an order by clause does not affect the ROWNUM of each row.

SEQUENCES

- Most application requires automatic generation of numeric value.
- A sequence is a database object that generates automatic sequence number for rows in tables.
- Oracle provides an object called a sequence that can generate numeric values.
- The value generated can have a maximum of 38 digits. A sequence can be defined to:
    - Generate numbers in ascending or descending.
    - Provide intervals between numbers
    - Caching of sequence numbers in memory etc.
- This information is provided to oracle at the time of sequence generation. The SQL statement used for creating sequence is:

```
Sql>CREATE SEQUENCE sequence_name
     [ INCREMENT BY integervalue
       START WITH integervalue
       MAXVALUE integervalue /   NOMAXVALUE
      MINVALUE integervalue  /   NOMINVALUE
      CYCLE/ NOCYCLE
      CACHE integervalue/ NOCACHE
      ORDER / NOORDER]
```

Sequence is always given a name so that it can be referenced later when required.

Example:
Create sequence order_seq, which will generate numbers from 1 to 9999 in ascending order with an interval of 1. The sequence must restart from the number 1 after generating number 9999.

```
Sql>CREATE SEQUENCE order_seq
     [ INCREMENT BY 1
       START WITH  1
       MINVALUE  1
      MAXVALUE   9999
    CYCLE;
```

Keywords	**Description**
INCREMENTED BY	Specifies the interval between sequence numbers. It can be positive or negative value but not zero. If this clause is omitted the default value is 1.
MINVALUE	Specifies the sequence minimum value.
MAXVALUE	Specifies the maximum value that the sequence can generate.

START WITH	Specifies the first sequence number to be generated. The default for a ascending sequence is the minimum value (1) and for a descending sequence, it is the maximum value (-1).
CYCLE	Specifies that the sequence continue to generate repeat values after reaching either its maximum value.
NO CYCLE	Specifies sequence cannot generate more values after reaching the maximum value.
CACHE	Specifies how many values of a sequence oracle pre allocate and keeps in memory for faster access. The minimum value for this parameter is two.
NOCACHE	Specifies that value of a sequence are not pre-allocated.

If the CACHE / NOCACHE clause is omitted oracle caches 20 sequence numbers by default. The ORDER / NOORDER clause has no significance, if oracle is configured with single server option. It is useful only when you are using Parallel Server in Parallel mode option.

Referencing a Sequence:

Once a sequence is created SQL can be used to view the values held in its caches. To simply view sequence value use a select sentence as described below.

```
Sql>SELECT sequence_name.nextval FROM dual;
```

This will display the next value held in the cache on the VDU screen. Every time **nextval** references a sequence its output is automatically incremented from the old value to the new value ready for use.

After creating a table you can add the data by using the INSERT command like this:

```
Sql>INSERT INTO sales_order(o_no, o_date, c_no)
            VALUES (order_seq.nextval, sysdate, 'c0001');
```

To references the current value of a sequence:

```
Sql>SELECT sequence_name.currval FROM dual;
```

Altering A Sequence

A sequence once created can be altered. This is achieved by using the ALTER SEQUENCE command.

```
Sql>ALTER SEQUENCE sequence_name
    [ INCREMENT BY integervalue
START WITH integervalue
 MAXVALUE integervalue /   NOMAXVALUE
MINVALUE integervalue  /    NOMINVALUE
CYCLE/ NOCYCLE
CACHE integervalue/ NOCACHE
ORDER / NOORDER]
```

Example:
Change the cache value of the sequence order_seq to 30 and interval between two numbers as 2.

```
Sql>ALTER SEQUENCE order_seq
      INCREMENTED BY 2
       CACHE 30;
```

Dropping A Sequence

The DROP SEQUENCE command is used to remove the sequence from the database.

```
Sql>DROP SEQUENCE sequence_name;
```

Views

- The table of a database defines the structure and the organization of its data. Once they are defined they always present the data in a particular way.
- After a table is created and populated with data, it may become necessary to prevent all users from accessing all columns of a table, for data security reason.
- This would mean creating several tables having the appropriate number of columns and assigning specific users to each table, as required.
- Sometimes in application, a different view of the data or the information in a different format is desired.
- This is handled in ORACLE using VIEWS.
- A VIEW is a virtual table in the database whose contents are defined by a query it can represent.
- A view holds no data at all, until a specific call to the view is made. This reduce redundant data on a HDD to a very large extent.

Characteristics of a view

- Define with any SELECT statement.
- Stored in the data dictionary in the table called USER VIEWS.
- Views do not exit physically. They are virtual tables that exit only as definitions record in the system catalogue.

Creation of views:

```
Sql>CREATE VIEW viewname AS
                         SELECT columnname, coluimnname

                         FROM tablename
                         WHERE columnname=expression list;
                         GROUP BY grouping criteria
                         HAVING predicate
```

The ORDER BY clause cannot be used while creating a view.

Example:
Create view on the client_master table for the Administration Department.

```
Sql>CREATE VIEW vw_clientadmin AS
                         SELECT  name, adddress1, address2, city, pincode, state
                         FROM client_master;
```

Selecting a data set from a view:
Once a view has been created, it can be queried exactly like a base table.

```
Sql>SELECT columnname, columnname
            FROM viewname;
```

The select statement can have the clause like WHERE, ORDER BY etc.

Example:

```
Sql>SELECT name, address1, address2, city, pincode, state
               FROM vw_clientadmin
               WHERE city IN ('Mehsana', 'Patan');
```

Destroying a view:

The DROP VIEW commnd is used to remove a view from the database.

```
Sql>DROP VIEW viewname;
```

Advantages of using views

Security:
Each user can be given permission to access the database only through a small set of views that contain the specific data.
The user is authorized to see, rather than the entire table, thus restricting the user's access to stored data.

Query Simplicity:

A view can draw data from several different tables and present it to the user as a single table, turning what would have been multi-table queries into simple single table queries against the view.

Insulation from changes:
A view can present a consistent, unchanged image of the structure of the database, even if the underlying tables are split, reconstructed or renamed.

Data integrity:
If the data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets specified integrity constraint.

Disadvantages of views

Performance:
View creates the appearance of a table, but the DBMS must still translate the queries against the view into the queries against the underlying source tables.
If the view is defined by a complex multi-join, then it may take long time to carry out.

Update restrictions:
More complex views can not be used to update the underlying source tables they are 'read only'.

GRANTING AND REVOKING PERMISSSIONS

- Objects that are created by a user are owned and controlled by that user.
- If a user wishes to access any of the objects belonging to another user, the owner of the object will have to give the permission for each access. This is called ***Granting of Privileges.***
- Privileges once given can be taken back by the owner of the object. This is called ***Revoking of Privileges.***

Granting Privileges using the GRANT statement.

The Grant statement provides various types of access to database objects such as tables, views and sequences.

```
Sql>GRANT object privileges
    ON object name
    TO username
     [WITH GRANT OPTION]
```

OBJECT PRIVILEGES:

Each object privileges that are granted authorize the grantee to perform some operation on the object. The user can grant the privileges or grant only specific object privileges.

The lists of object privileges are as follows:

ALTER	: allows the grantee to change the table definition with the ALTER TABLE command.
DELETE	: allows the grantee to remove the record from a table with the DELETE Command.
INDEX	: allows the grantee to create an index on the table with the INDEX Command.
INSERT	: allows the grantee to add records to the table with the INSERT Command.
SELECT	: allows the grantee to query the table with the SELECT Command.
UPDATE	: allows the grantee to modify the records in the table with the UPDATE Command.

WITH GRANT OPTION:

The WITH GRANT OPTION allows the grantee to in turn grant object privileges to other users.

Example:
Give the user Pradeep all data manipulation permission on the table product_master.

```
Sql>GRANT ALL
              ON product_master
               TO pradeep;
```

Give the user Mita only the permission to view and modify records in the table client_master.

```
Sql>GRANT select, update
             ON client_master
              TO Mita;
```

Revoking Privileges Given:
Privileges once given can be denied to a user using the REVOKE command.

```
Sql>REVOKE object privileges
     ON object name
      FROM username
```

Example:
Take back all privileges on the table bonous from Florian.

```
Sql>REVOKE ALL
               ON bonous
               FROM Florian;
```

TRANSCATION PROCESSSING IN SQL

- ORACLE allows the grouping of individual changes done through the use of SQL commands like SELECT, INSERT, UPDATE or DELETE, INTO logical transactions to ensure data consistency and integrity.
- Transaction is a logical unit of work or a sequence of logical operations or event which is treated as a single unit. In other words, transaction is a series of SQL statements that either succeed or fail as a unit.
- When you manipulating the data in the database, the changes made by you are not normally made permanent in the database until you give a command to do so or you exit oracle.

Saving the changes in the transaction

You can make changes permanent in the database by using.

Sql> COMMIT [WORK];

WORK is optional and only provided for readability.
Following things happen when you issue COMMIT.

- The changes are permanent in the database.
- The transaction is marked as closed.
- New transaction is started.
- Any locks acquired by the transaction are released.

Undo the changes made in the transaction

- It may so happen that you may not want to make the data permanent in the database, due to the failing of some SQL statement in a sequence or wrong information provided; then you may use the command ROLLABACK to discard the change made.

Sql>ROLLBACK [WORK];

Following things happen when you use ROLLBACK:

- Any changes made by the transaction are undone, as if it hadn't been issued.
- New transaction is started.
- Any locks acquired by the transaction are released.

ROLLBACK statement undoes the entire transaction. If you want to undo only part of the transaction, then it can be done by using

Sql>SAVEPOINT name;

During the transaction if you want to undo the changes up to certain savepoint, you have to give the ROLLBACK command like…

Sql>ROLLBACK [WORK] TO SAVEPOINT name;

Following things happen when you issue ROLLBACK TO SAVEPOINT.

- The changes made by the statement issued after the savepoint, are undone. The savepoint remains active, so it can be rolled back to again if desired.
- Any locks and resources acquired by the SQL statements since the save point are released.
- The transaction is still active as there are SQL statements pending.
- Savepoint are useful in interactive programs, because you can create and name intermediate steps of a program. This allows you more control over longer, more complex program.

Introduction to PL/SQL:-

PL/SQL stands for Procedural Language/Structured Query Language. It is standard procedural language that helps user by combining the power and flexibility of SQL with various procedural construct.

A PL/SQL segment is called a block. A block consists of three sections. These sections are:
1) Declare section.
2) Begin section.
3) Exception section.
4) End section.

Declare section :-

Code block start with a declaration section, in which memory variable and other Oracle object can be declared, and if required initialized. Once declared,they can be used in the SQL statement for data manipulation.

Begin section:-

It consists of a set of SQL and PL/SQL statements, which describe processes that have to be applied to data. Actual data manipulation, retrieval, looping and branching construct are specified in this section.

Exception section:-

This section deals with handling of errors that arise during execution of the data manipulation statements which make up the PL/SQL code block. Errors can arise due to syntax, logic and/or validation rule violation.

End section:-

This mark the end of a PL/SQL block.

Block Structure:-

```
DECLARE
   <Declaration of memory variable, constants, cursors>
 BEGIN
    <Executable statement>
EXCEPTION
    <Exception handlers>
END;
```

PL/SQL Control Structure:-

PL/SQL has a variety of control structures that allow you to control the behavior of the block as it runs.

IF-THEN-ELSE statements

```
   IF  <Boolean Expression>  Then
        Sequence of statements;
  ELSIF<Boolean Expression> Then
        Sequence of statements;
  ELSE
       Sequence of statements;
  END IF;
```

LOOPS:-

```
   LOOP
       Sequence of statement;
       EXIT [WHEN condition];
   END LOOP;

   WHILE<condition>LOOP
       Sequence of statement;
   END LOOP;

   FOR<variable>IN<lower value>..<Upper value>LOOP
       Sequence of statement;
```

```
    END LOOP;
```

PL/SQL Data Types:-

The default data types that can be declared in PL/SQL are Number, Char, and Varchar2, Boolean, date.

%TYPE attribute is used when you want a variable to have the same data type as that of column of a table. Using %type has the advantage that whenever the type and/or
Size of a column in the table is changed; it is automatically reflected in the variable. For example the following statement declares variable **NAME** to have the same data type as that of the column **EMP_NAME** of **EMPLOYEE** table.

```
NAME     EMLOYEE.EMP_NAME%TYPE;
```

%ROWTYPE attribute is used when you want the variable to hold the values of an entire row of a table. In case variables for the entire row of a table need to be declared, the instead of declaring them individually, **%ROWTYPE** should be used in following manner.

```
EMP_ROW        EMPLOYEE%ROWTYPE
```

The above statement declares **EMP_ROW** variable as composite variable consisting of the column names of the **EMPLOYEE** table as its members. It is capable of storing one whole row of a table. To refer to a value in the row, say **ENAME** a **(.)** followed by the column name is used as shown below:

```
EMP_ROW.ENAME:='ABC';
```

Advantages of PL/SQL:

1. PL/SQL is development tool that not only supports SQL data manipulation but also provides facilities of conditional checking, branching and looping.

2. PL/SQL sends an entire block of statements to the Oracle engine at one time. The communication between the program block and the Oracle engine reduces considerably. This in turn reduces network traffic. The Oracle engine gets the SQL statements as a single block, and hence it processes this code much faster than if it got the code one sentence at a time.

3. PL/SQL also permits dealing with errors as required, and facilitates displaying user-friendly messages, when errors are encountered.

4. PL/SQL allows declaration and use of variable in block of code. These variables can be used to store intermediate results of a query for faster processing, or calculate values and insert them into an Oracle table.

5. Application written in PL/SQL is portable to any computer hardware and operating system, where oracle is operational.

Program:

1. Write a PL/SQL code to print numbers 1 to 8. Also insert the result into a table TEMP, having one column of NUMBER data type.
2. Write PL/SQL code to calculate the area of a circle for a value of radius varying from 3 to 7. Store the radius and the corresponding values of calculated area in an empty table named AREA, consisting of two columns radius and area.
3. Write a PL/SQL block to find the factorial of a given number.
4. Write a PL/SQL code to insert all the details of the employee whose code is 107 to a new table EMP which has same structure as EMPLOYEE table.
5. Write a PL/SQL code block that will accept an account number from the user, check if the user's balance is less than the minimum balance, only then deduct Rs. 100/- from the balance. The process is fired on the ACC_MSTR table.

Cursor:-

- The Oracle engine uses a work area for its internal processing in order to execute an SQL statement. This work area is private to SQL's operation and is called as Cursor. The SQL statement are executed in this work area and Query results are also stored in this work area.
- The date that is stored in the cursor is called the Active Data set.
- When cursor is loaded with multiple rows via query the Oracle engine open and maintains a row pointer. Depending on user request to view data the row pointer will be relocated within the cursor's Active Data Set.
- The Oracle also maintains the cursor variables and the values held in these variables indicate the status of the processing being done by the cursor.
- Depending upon the circumstances under which they are opened there are two types of cursor. 1) Implicit Cursor 2) Explicit Cursor

Implicit cursor: -

- The oracle engine implicitly opens a cursor on the server to process each SQL statement.
- Implicit cursor is opened and managed by the Oracle engine internally, the function of reserving an area in memory, populating this area with appropriate data, processing the data in memory area, releasing the memory area when the processing is complete is taken care of by the Oracle engine.
- The resultant data is then passed to the client via network.
- Implicit cursor attributes can be used to access information about the status of last insert ,update , delete or single row select statement.

Implicit cursor Attributes: -

- **%ISOPEN**: - The Oracle engine automatically opens and close the SQL cursor after executing its associated select, insert, delete and update SQL statement.

- **%FOUND: -** Evaluate to TRUE, if an insert, update or delete affected one or more Rows, or a single-row select returned one or more rows. Otherwise it evaluates to FALSE.

- **%NOTFOUND: -** It is the logical opposite of %FOUND. It evaluates to TRUE, if an insert, update or delete affected no rows, or a single-row select returns no rows. Otherwise it evaluate to FALSE.

- **%ROWCOUNT: -** Returns the number of rows affected by an insert, update or delete or select into statement.

Explicit Cursor: -

Explicit cursors are user defined cursor. The cursor is opened and managed by the user. This cursor will be declared and mapped to an SQL query in the Declare section of the PL/SQL block and used within the Executable Section.

Cursor Management:-

- Declare a cursor mapped to a SQL select statement that retrieves data for processing.
- Open cursor
- Fetch data from the cursor one row at a time into memory variables.
- Process the data held in the memory variables as required using aloop.
- Exit from the loop after processing is complete.
- Close the cursor.

Handling of Cursor:-

A cursor is defined in the declarative part of PL/SQL block. This is done by naming the cursor and mapping it to a query.
Syntax:

```
CURSOR cursorname IS
    SQL select statement;
```

The three statements that are used to control the cursor are open, fetch and close.

Open statement:

- Opening a cursor execute the query and creates the active set that contains all rows, which meet the query search criteria.
- An Open statement retrieves records from a database table and places the records in the cursor.
- A cursor is opened in server memory.
- Syntax:
  ```
  OPEN cursorname;
  ```

Fetch Statement:

- The Fetch statement retrieves the rows from the active set opened in the server into memory variables declared in the PL/SQL code block on the client one row at a time. The memory variables are opened on the client machine.
- Each time a fetch is executed, the cursor pointer is advanced to the next row in the Active Data Set.
- A standard loop structure (Loop….End loop) is used to fetch records from the cursor into memory variables one row at a time.
- Syntax:

  FETCH cursorname INTO variable1, variable2,….;

Close statement:

- The close statement disables the cursor and the active date set becomes undefined.
- This will release the memory occupied by the cursor and data set both on the Client and on the Server.
- Syntax:

  CLOSE cursorname;

Explicit Cursor Attributes: -

- **%ISOPEN: -** Evaluate to TRUE, if an explicit cursor is open, or to FALSE, if it is closed.

  Cursorname%ISOPEN.

- **%FOUND: -** Evaluate to TRUE, if the last fetch succeeded because a row was available, or to FALSE, if the last fetch failed because no more rows were available.

  Cursorname%FOUND.

- **%NOTFOUND: -** Evaluate to TRUE, if the last fetch has failed because no more rows were available, or to FALSE, if the last fetch returned a row.

  Cursorname%NOTFOUND.

- **%ROWCOUNT: -** Returns the number of rows fetched from the active data set. It set to zero, when the cursor is opened.

  Cursorname%ROWCOUNT.

Parameterized Cursor

Commercial application require that the query, which, defines the cursor, be generic and the data that is retrieved from the table be allowed to change according to need.

Oracle recognizes this and permits the creation of a parameterized cursor prior opening. Hence the content of the opened cursor will constantly change depending upon a value passed.

Since the cursor accepts values or parameter it is called as ***Parameterized Cursor.*** The parameters can be either a constant or a variable.

Declaring a Parameterized Cursor:

Syntax:

CURSOR cursor_name(variable_name data type) is <**SELECT** statement…>

Opening a Parameterized cursor and passing values to the cursor:

Syntax:

OPEN cursor_name(value/variable/expression)

Program:

1. Write PL/SQL code that uses a cursor to list the four highest paid employees from the employee table.
2. Write a PL/SQL code that retrieve a number from two tables, then insert the sum of the numbers into a third table. It stops when all rows have been retrieved from both tables.
3. Write a PL/SQL code that using a cursor that display the details of those employees from employee table who have joined after '01-jan-07'.

All The Best
Team C2C